

INDEX

S.No	Topic	Page No.
1	Abstract	4
2	Introduction	4
3	Machine learning Models	4
4	Our Approach	5
5	Data Description	6
6	Visualizing the Data	6
7	Computing the Cost	8
8	Using Gradient Descent	9
9	Plotting the hypothesis	12
10	Prediction	13
11	Visualizing the cost function	14
12	Conclusion	15

ABSTRACT: The estimation of reliability and prediction of failure are some of the very important aspect for many production companies. The main goal of this report is to make this calculative process a bit easier than conventional methods by using latest machine learning techniques. The project is a step by step procedure that explains each and every aspect of how a machine learning model can help predicting failure frequency. Starting with illustrating the given data using 2D graphs, the algorithm and code have been explained at each forgoing step.

INTRODUCTION:

With growing international markets and globalization, there has also been a subsequent increase in global competition for manufacturing companies. Now, they have to produce high quality products in much lesser time so as to satisfy the customer needs. This arises a major need for reliability and today when most of the complex work is being done by computers why not this. We all agree with the fact that when it comes to larger & complex mathematics computers are by far superior to humans because of their high efficiency, higher rate of accuracy and quick delivery of results.

The frequency of failure plays a major role in determining the reliability of a product. If we develop a tool that predicts this frequency for any given time, it would get much easier for the manufacturers to check errors, process them and deliver much better and reliable products to their consumers.

MACHINE LEARNING METHODS:

According to one of the oldest definitions of Machine Learning, it is defined as the field of study that gives computers the ability to learn without being explicitly programmed. In general almost all the machine learning problems can be divided into two categories as follows:

1. Supervised Learning: In such problems we are given a dataset and through that we can calculate the relation between input and output which gives us an idea of how to predict output for some other future inputs.
Supervised Learning Problems are further divided into two categories:
 - a. Regression: When input and output have almost continuous relationship and can be represented successfully on a graph.
 - b. Classification: In this case, we try to predict the results as a discrete set of outputs like classifying if a given image is of car or bike when some images and their category has been given to us in the past.
2. Unsupervised Learning: In such problems no primary input and output relationship is specified. In this technique we have to approach problems with little or no idea about what our results should look like. An eg. Of this method is the clustering problem.

OUR APPROACH:

When we analyzed the failure frequency v/s time data we found that the data can be successfully represented on a 2d graph as a continuous relationship between the two properties. Thus, we decided to go for linear regression supervised machine learning model and implemented it to predict the failure frequency as other different times.

DATA DESCRIPTION:

So, for keeping the analysis easier we have used just two columns of data. The first column is the “**Number of components failed**” and the second column describes the respective “**Number of days**”. The electronic component studied for reliability management is a 12 V battery whose lifetime is approximately six to seven years. A total of 79,154 batteries were produced by a certain company whose dataset of failure v/s time has been recorded. The data set has 97 rows and 2 columns. Now, on the basis of this data provided, we fit a linear regression machine learning model to the data so as to predict the number of batteries that would fail at any given time ‘t’.

***NOTE:** For keeping the calculation in the minimum ranges, both the column values have been divided by 100 while the learning model is trained over the dataset.*

TRAINING THE ML MODEL:

STEP1: Visualizing the Data

Before starting any task during the development of Machine Learning Model, the first step should always be plotting and visualizing the dataset because it is useful to understand the data and it gives us an idea of what our data actually looks like.

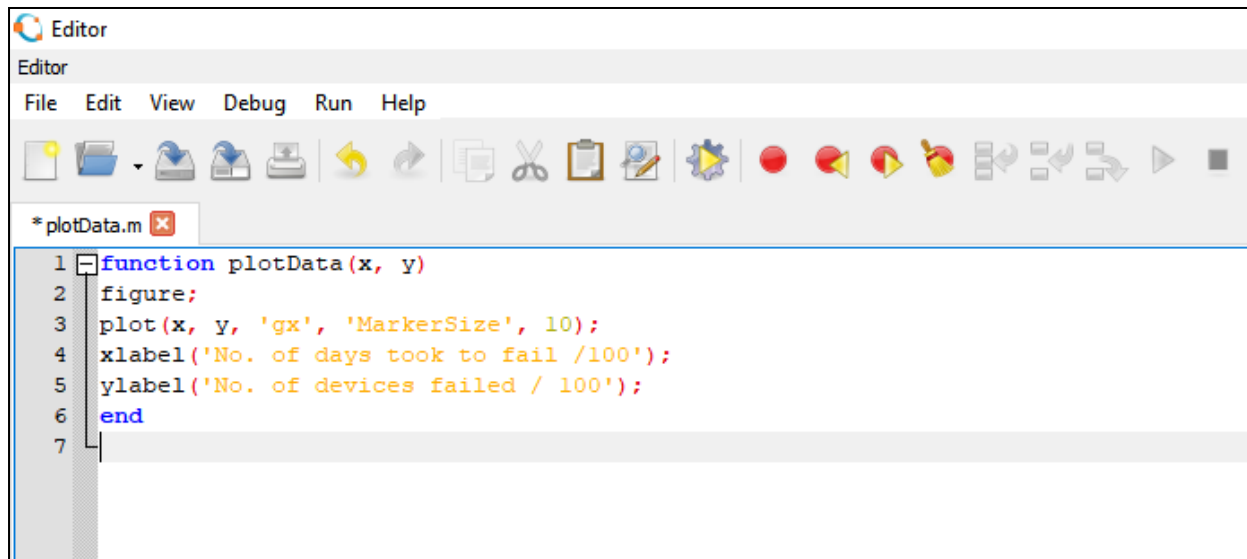
So, the dataset we are using has only two properties:

Coloumn 1: Number of components failed

Coloulm 2: Number of days

So, a 2d x-y plot would work well.

For plotting the data we have used plotdata.m function. Below is the screenshot of the code inside the plot data function. This function when compiled gives the following output plot between the coloumns of out dataset.



```
1 function plotData(x, y)
2 figure;
3 plot(x, y, 'gx', 'MarkerSize', 10);
4 xlabel('No. of days took to fail /100');
5 ylabel('No. of devices failed / 100');
6 end
7
```

Figure 1: Screenshot of the Code to plot the training dataset

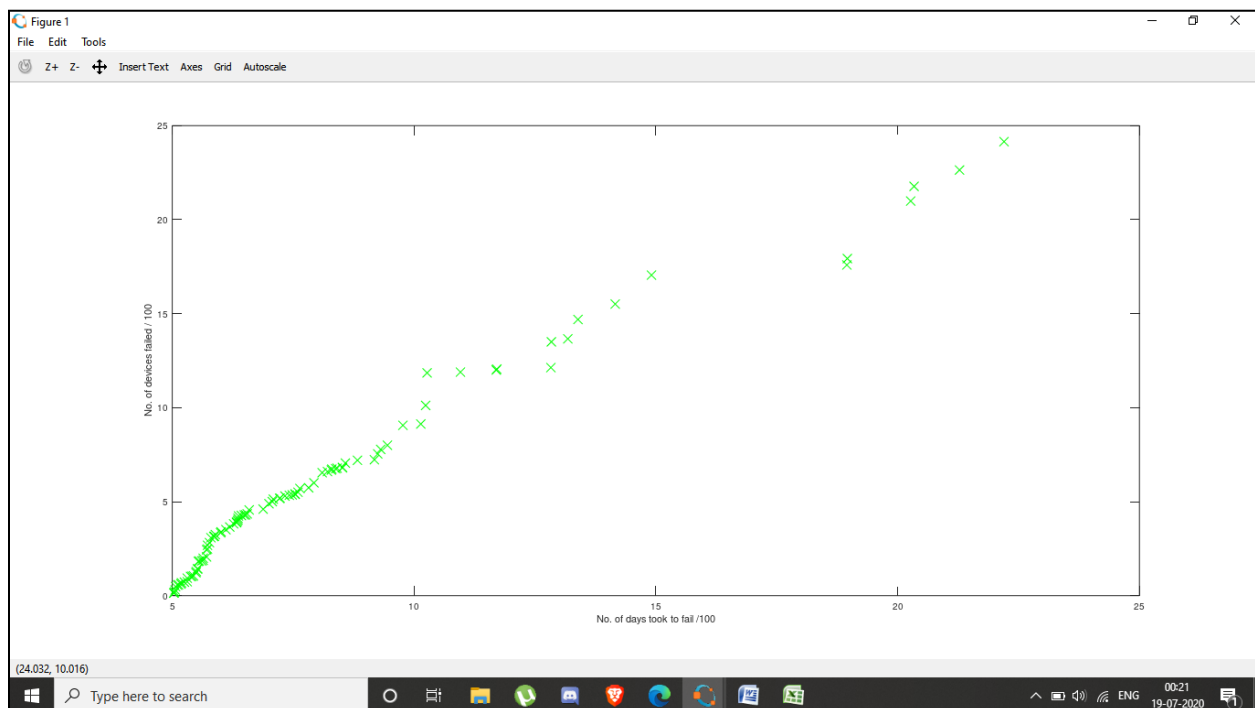


Figure 2: Scatter Plot of Training Data

STEP2: Computing the cost

Whenever we implement any machine learning model, like linear regression, we try to fit a straight line the dataset and the equation representing this straight line is called the hypothesis function. Our hypothesis function given by $h(x)$ is of the form:

$$h(x) = \Theta_0 + \Theta_1 x$$

where Θ_0 and Θ_1 are called the parameters, x represent the data on x axis or the input training example in this case “Number of days”, and y represents the y axis or the output training example i.e. in this case “Number of components failed”.

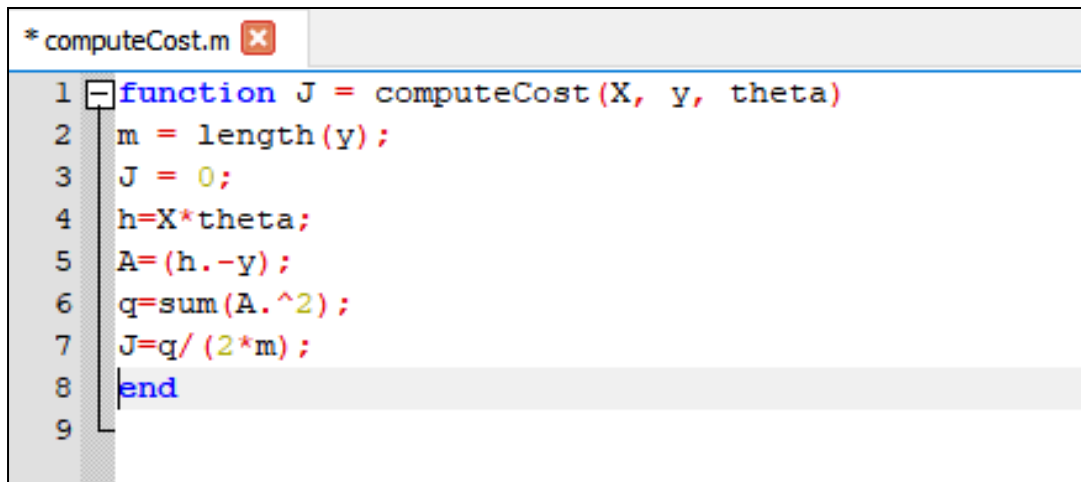
How well our proposed hypothesis function or straight line fits the dataset, its measure is given by the cost function called $J(\Theta_0, \Theta_1)$.

The cost function is mathematically given by:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2$$

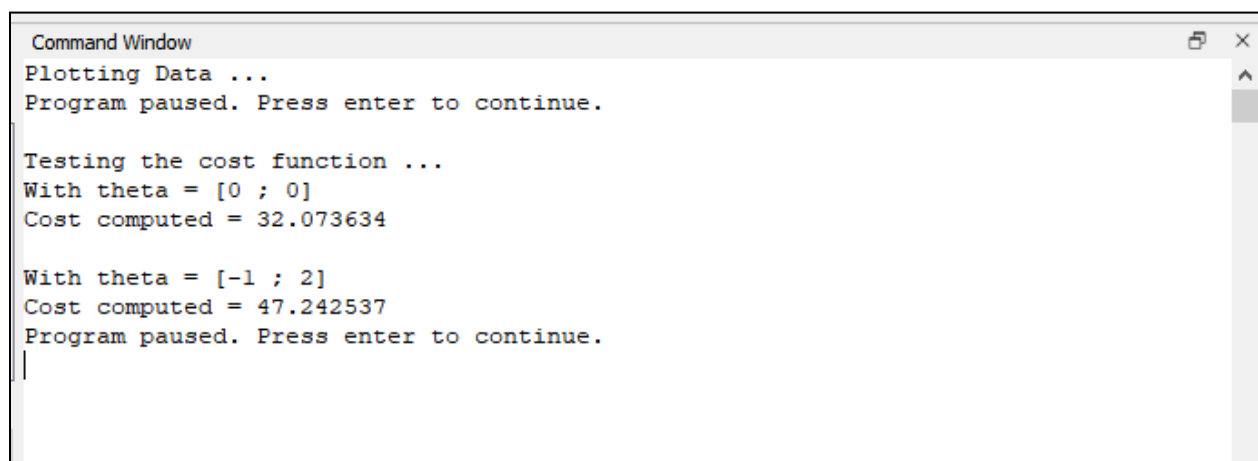
So, our main aim in the whole model is to find some value of Θ_0 and Θ_1 , such that we have minimum value of cost function which means our straight line fits the data more perfectly.

The code for computing the cost function is written in the function `compute cost` and is shown below. The code gives the output cost computed over the given dataset when provided with any random values of parameters $[\Theta_0, \Theta_1]$.



```
* computeCost.m
1 function J = computeCost(X, y, theta)
2 m = length(y);
3 J = 0;
4 h=X*theta;
5 A=(h.-y);
6 q=sum(A.^2);
7 J=q/(2*m);
8 end
9
```

Figure 3: Screenshot of the Code to compute the cost function



```
Command Window
Plotting Data ...
Program paused. Press enter to continue.

Testing the cost function ...
With theta = [0 ; 0]
Cost computed = 32.073634

With theta = [-1 ; 2]
Cost computed = 47.242537
Program paused. Press enter to continue.
```

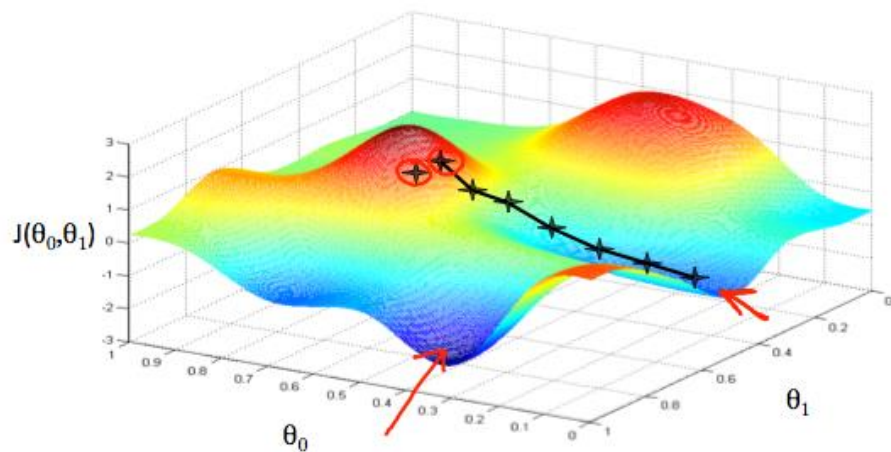
Figure 3: Output – Computed Cost Function for given parameter values

STEP3: Using Gradient Descent to minimize the Cost Function

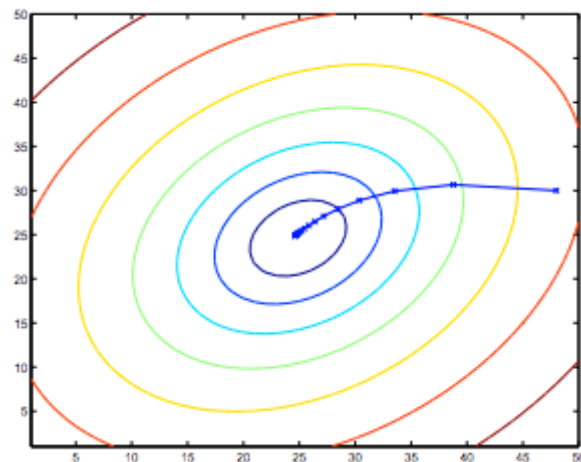
Now, we use a complex mathematical algorithm called “THE GRADIENT DESCENT” to minimize our cost function. This algorithm basically reaches to the minima of the cost function through subsequent iterations. Mathematically, Gradient Descent is given by:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (\text{simultaneously update } \theta_j \text{ for all } j).$$

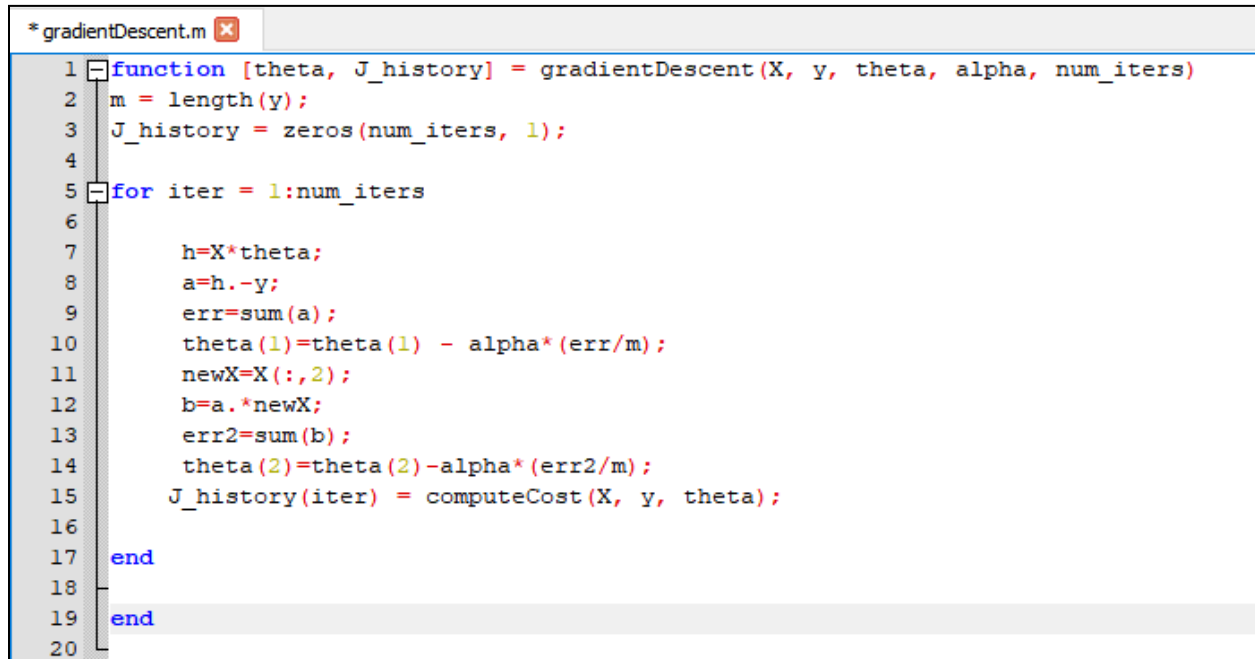
For eg. If the cost function has a 3D plot as shown, then gradient descent goes step by step and finally finds us the lowest point in the graph thereby giving the minimum value.



If a 3D plot of cost function wrt to the parameters is drawn in the form of a contour plot, the gradient descent algorithm finds the minimum value which is located at the centre of the contour plot as shown.

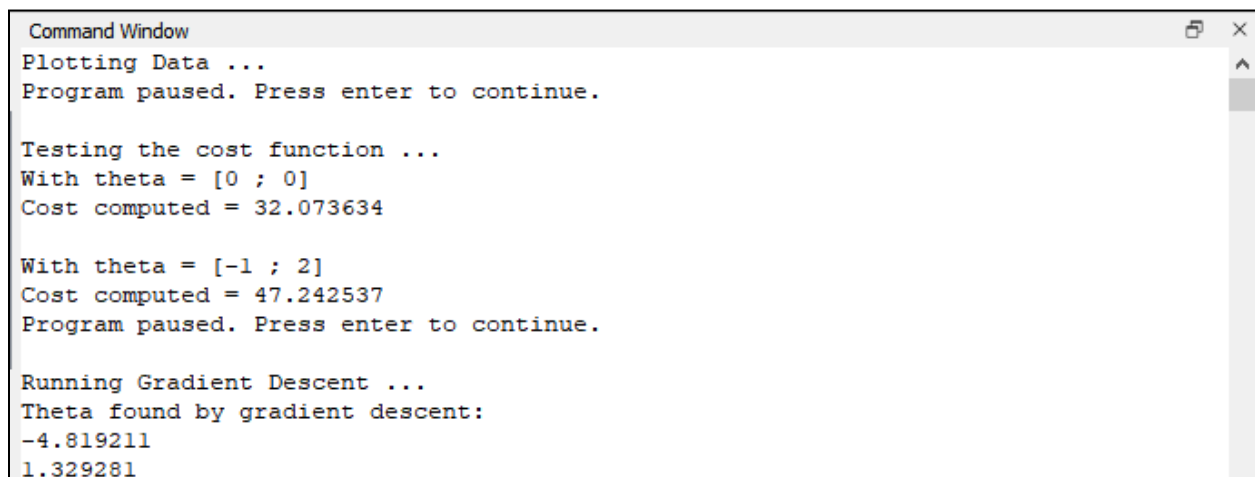


The code for Gradient Descent Algorithm has been written in gradientDescent.m function and it returns the value of parameters Θ_0 and Θ_1 , for which cost function would be minimum that is for which hypothesis would fit the training set well.



```
* gradientDescent.m
1 function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
2 m = length(y);
3 J_history = zeros(num_iters, 1);
4
5 for iter = 1:num_iters
6
7     h=X*theta;
8     a=h.-y;
9     err=sum(a);
10    theta(1)=theta(1) - alpha*(err/m);
11    newX=X(:,2);
12    b=a.*newX;
13    err2=sum(b);
14    theta(2)=theta(2)-alpha*(err2/m);
15    J_history(iter) = computeCost(X, y, theta);
16
17 end
18
19 end
20
```

Figure 4: Code for implementing Gradient Descent



```
Command Window
Plotting Data ...
Program paused. Press enter to continue.

Testing the cost function ...
With theta = [0 ; 0]
Cost computed = 32.073634

With theta = [-1 ; 2]
Cost computed = 47.242537
Program paused. Press enter to continue.

Running Gradient Descent ...
Theta found by gradient descent:
-4.819211
1.329281
```

Figure 5: Output – Returning the value of parameters

STEP4: Plotting the hypothesis function wrt given dataset

Now, since we have calculated the required parameters we can rewrite our hypothesis function as :

$$h(x) = (-4.819211) + (1.329281) x$$

Now, using this calculated hypothesis function we try to fit a straight line to our training set using the following code:

```
41 % Plot the linear fit
42 hold on;
43 plot(X(:,2), X*theta, '-')
44 legend('Training data', 'Linear regression')
45 hold off
46
```

Figure 6: Code for plotting the hypothesis function with calculated theta values

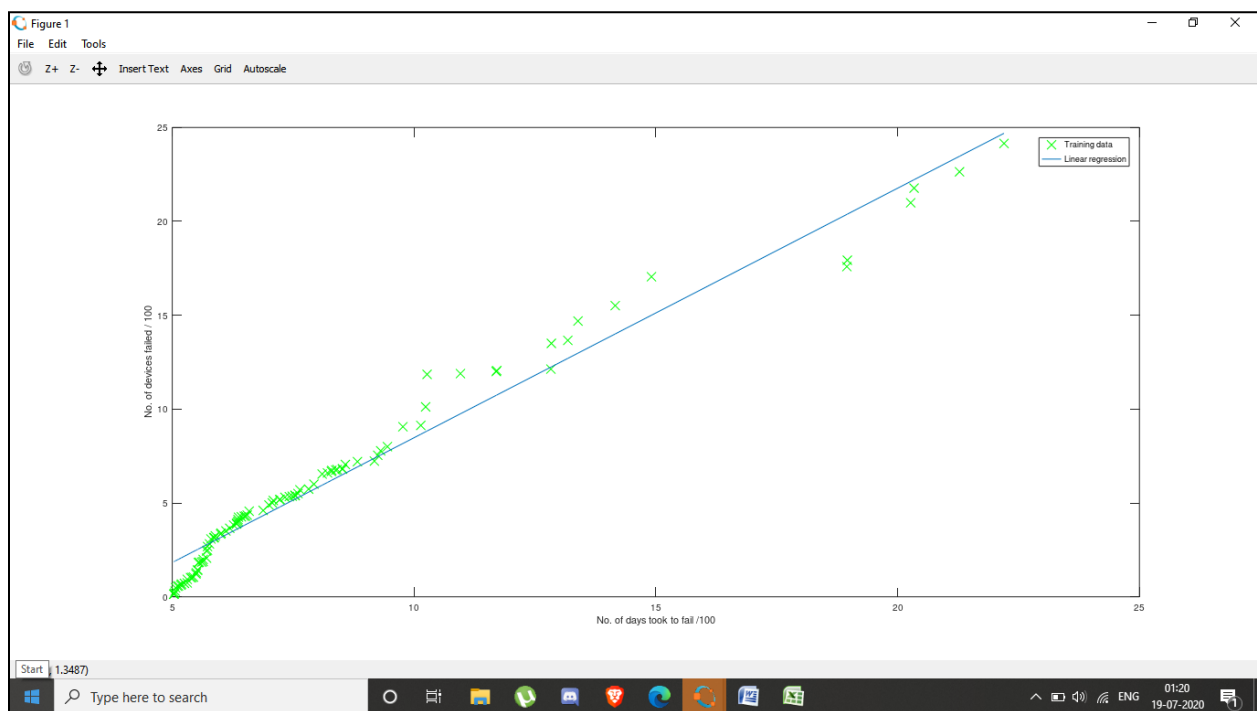


Figure 7: Linear hypothesis function (blue line) over the training set (green crosses)

STEP4: Prediction

By now, we have calculated our hypothesis function $h(x)$ and using it we can successfully predict values for any given 'x' or Number of components failed for given Number of days.

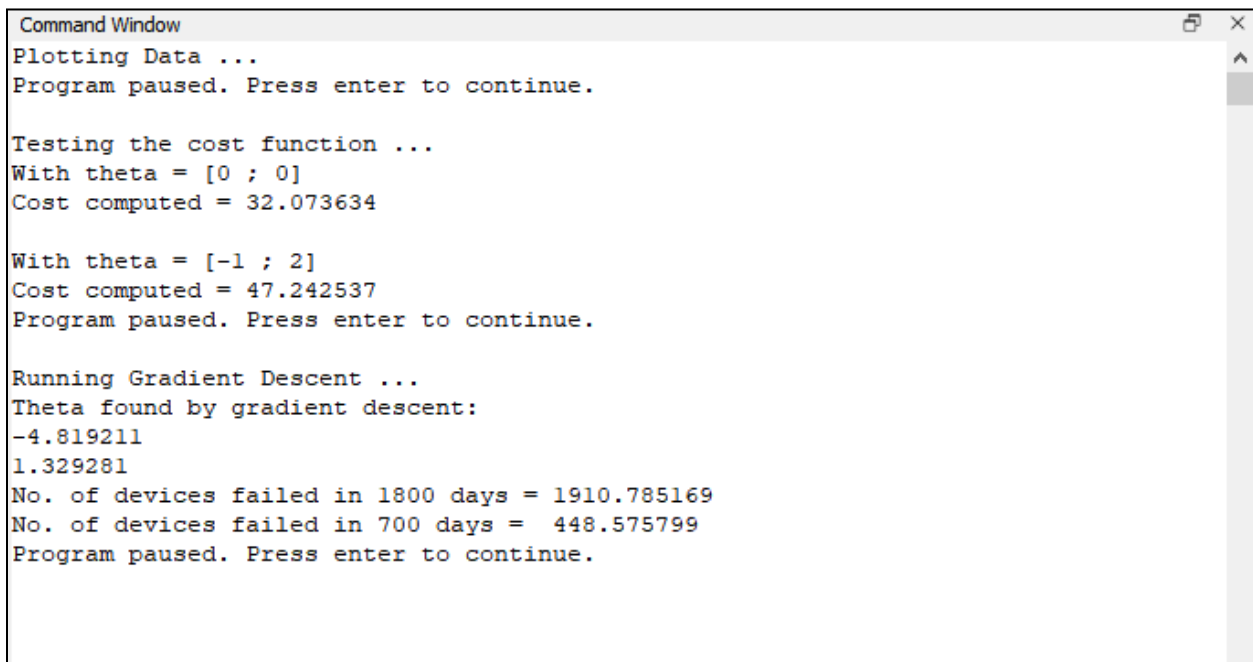
Suppose we want to calculate number of components failed for 700 and 1800 days, we do so by writing just a few lines of code now.

```

47 % Predict values for population sizes of 35,000 and 70,000
48 predict1 = [1, 18] * theta;
49 fprintf('No. of devices failed in 1800 days = %f \n', ...
50     predict1*100);
51 predict2 = [1, 7] * theta;
52 fprintf('No. of devices failed in 700 days = %f \n', ...
53     predict2*100);
54
55 fprintf('Program paused. Press enter to continue.\n');
56 pause;

```

Figure 8: Code for predicting no. of components failed for a given no. of days



```

Command Window
Plotting Data ...
Program paused. Press enter to continue.

Testing the cost function ...
With theta = [0 ; 0]
Cost computed = 32.073634

With theta = [-1 ; 2]
Cost computed = 47.242537
Program paused. Press enter to continue.

Running Gradient Descent ...
Theta found by gradient descent:
-4.819211
1.329281
No. of devices failed in 1800 days = 1910.785169
No. of devices failed in 700 days = 448.575799
Program paused. Press enter to continue.

```

Figure 9: Output

STEP5: Visualizing the cost function

As stated earlier, 3D and 2D (contour plot) plots can be used to visualize the cost function $J(\Theta_0, \Theta_1)$ and it can help us to see how Gradient Descent manages to calculate the minimum value of parameters Θ_0 and Θ_1 .

Below is the code that shows how to plot the cost function in two different forms which are as follows:

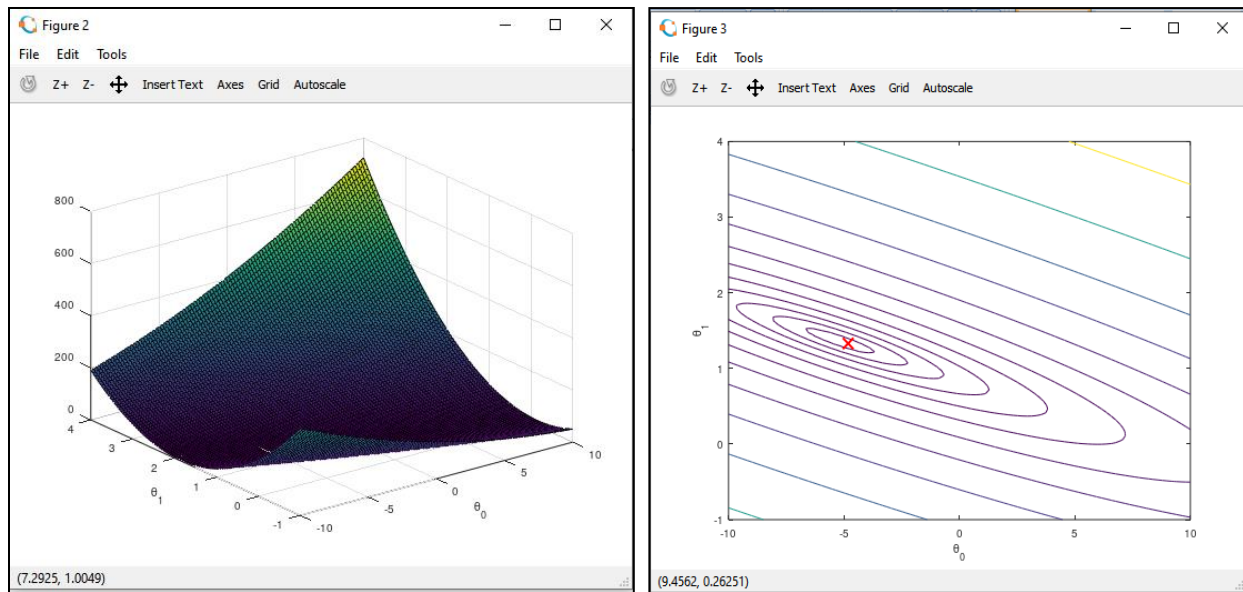
```

59 fprintf('Visualizing J(theta_0, theta_1) ...\n')
60
61 theta0_vals = linspace(-10, 10, 100);
62 theta1_vals = linspace(-1, 4, 100);
63 J_vals = zeros(length(theta0_vals), length(theta1_vals));
64
65 for i = 1:length(theta0_vals)
66     for j = 1:length(theta1_vals)
67         t = [theta0_vals(i); theta1_vals(j)];
68         J_vals(i,j) = computeCost(X, y, t);
69     end
70 end
71
72 J_vals = J_vals';
73 % Surface plot
74 figure;
75 surf(theta0_vals, theta1_vals, J_vals)
76 xlabel('\theta_0'); ylabel('\theta_1');
77 % Contour plot
78 figure;
79 % Plot J_vals as 15 contours spaced logarithmically between 0.01 and 100
80 contour(theta0_vals, theta1_vals, J_vals, logspace(-2, 3, 20))
81 xlabel('\theta_0'); ylabel('\theta_1');
82 hold on;
83 plot(theta(1), theta(2), 'rx', 'MarkerSize', 10, 'LineWidth', 2);
84

```

line: 83 | col: 66 | encoding: SYSTEM | eol: LF

Figure 9: Code for plotting surface plot and contour plots of $J(\Theta_0, \Theta_1)$



*Figure 10: a. surface plot and b. contour plot
of $J(\Theta_0, \Theta_1)$*

CONCLUSION:

This project report successfully demonstrates the use of linear regression machine learning algorithm in predicting the component failure frequency for a given no. of day's time.

The model can be really helpful for large scale production companies to access flaws in the component and research the reason behind the untimely failures so as to provide their consumers with best quality products.

Also, such a model can be of great importance to buyers and even companies which might use this hardware component as a base for their device to know the reliability of the component they are purchasing so that they can have maximum profit.

One crucial drawback of this machine learning model is that it has not been trained on the components which have not failed until the recorded dataset. Hence, this model does not assure the accuracy of prediction of failure frequency

when it comes to improved revisions or other models of the same hardware component.

To overcome this particular drawback, we can train and test the model on another dataset recorded in future which would contain information about failure frequency of improved versions of the component.